



本科生实验报告

实验课程：____操作系统原理实验____

实验名称：____编译内核/利用已有内核创建 OS____

专业名称：____计算机科学与技术____

学生姓名：____张玉瑶____

学生学号：____23336316____

实验地点：____实验楼 B203____

实验成绩：____

报告时间：____2025 年 4 月 3 日____

Section 1 实验概述

- 实验任务 1： 复现 Example 1，使用 LBA28 的方式读取硬盘。再修改代码，利用 CHS 方式读取硬盘。
- 实验任务 2： 复现进入保护模式的操作
- 实验任务 3： 将 Lab2-Assignment4 的代码加载到保护模式

Section 2 实验步骤与实验结果

----- 实验任务 1 -----

- 任务要求：
 1. 复现 example1，使用 LBA28 的方式读取硬盘。这个方式给出逻辑扇区号即可，但需要手动去读取 I/O 端口。
 2. 修改代码，利用 CHS 读取硬盘。这个方式利用 BIOS 在实模式下读取硬盘的中断，其不需要关心具体的 I/O 端口，只需要给出逻辑扇区号对应的磁头（Heads）、扇区（Sectors）和柱面（Cylinder）即可。需要给出逻辑扇区号向 CHS 的转换公式
- 思路分析：对于 1.2，在 example1 的基础上修改代码，保持原有流程，添加转换公式。
 - 1) 删除 LBA28 端口操作代码；
 - 2) 添加 LBA→CHS 转换公式
 - 3) 使用 INT 13h AH=02h 中断实现 CHS 模式读取。
- 实验步骤：

example1:

 - 1) 将代码复制入 bootloader.asm 和 mbr.asm
 - 2) 编译运行

```

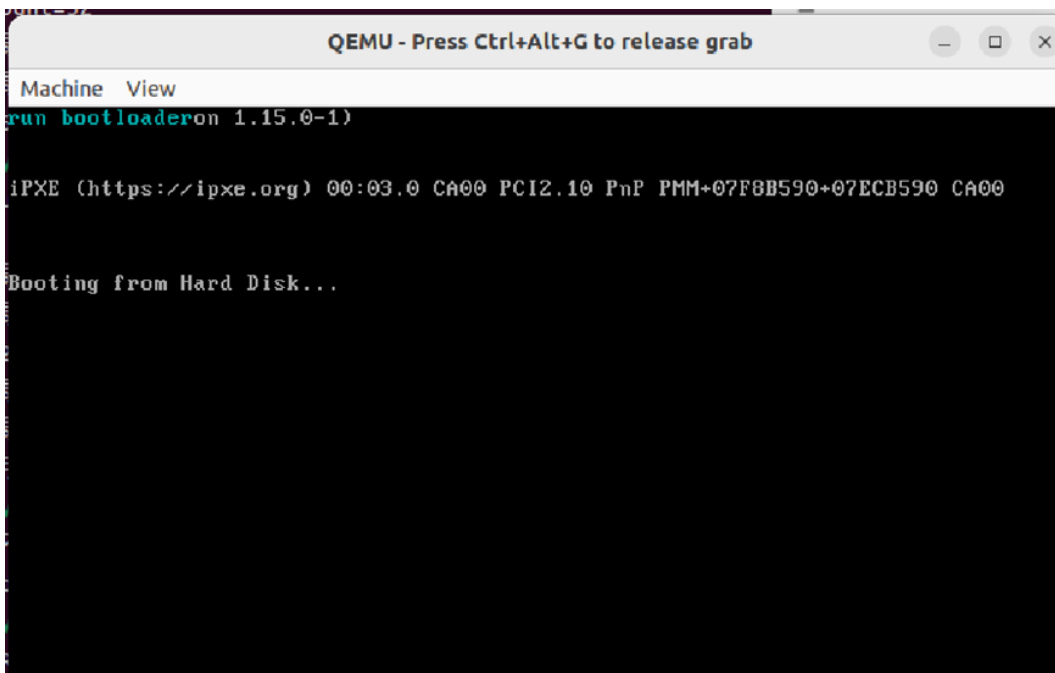
zyy@VirtualBox:~/lab1/src$ nasm -f bin mbr.asm -o mbr.bin
nasm -f bin bootloader.asm -o bootloader.bin
zyy@VirtualBox:~/lab1/src$ dd if=/dev/zero of=hd.img bs=1M count=32
记录了32+0 的读入
记录了32+0 的写出
33554432字节 (34 MB, 32 MiB) 已复制, 0.0454946 s, 738 MB/s
zyy@VirtualBox:~/lab1/src$ dd if=mbr.bin of=hd.img conv=notrunc
dd if=bootloader.bin of=hd.img bs=512 seek=1 conv=notrunc
记录了1+0 的读入
记录了1+0 的写出
512字节已复制, 0.00753361 s, 68.0 kB/s
记录了0+1 的读入
记录了0+1 的写出
52字节已复制, 0.000119595 s, 435 kB/s

```

```

zyy@VirtualBox:~/lab1/src$ qemu-system-x86_64 -hda hd.img
WARNING: Image format
was not specified for 'hd.img' and probing guessed raw.
       Automatically detecting the format is dangerous for raw images, write o
perations on block 0 will be restricted.
       Specify the 'raw' format explicitly to remove the restrictions.

```



1.2:

1) 写出转换公式:

$$\text{柱面号}(C) = \text{LBA} / (\text{HPC} \times \text{SPT})$$

$$\text{磁头号}(H) = (\text{LBA} / \text{SPT}) \% \text{HPC}$$

$$\text{扇区号}(S) = (\text{LBA} \% \text{SPT}) + 1$$

2) 在 mbr.asm 修改, 添加转换公式, 添加中断。

```

; 调用BIOS读取扇区
mov ah, 0x02      ; 功能号
mov al, 1         ; 读1个扇区
int 0x13
jc $              ; 出错则死循环

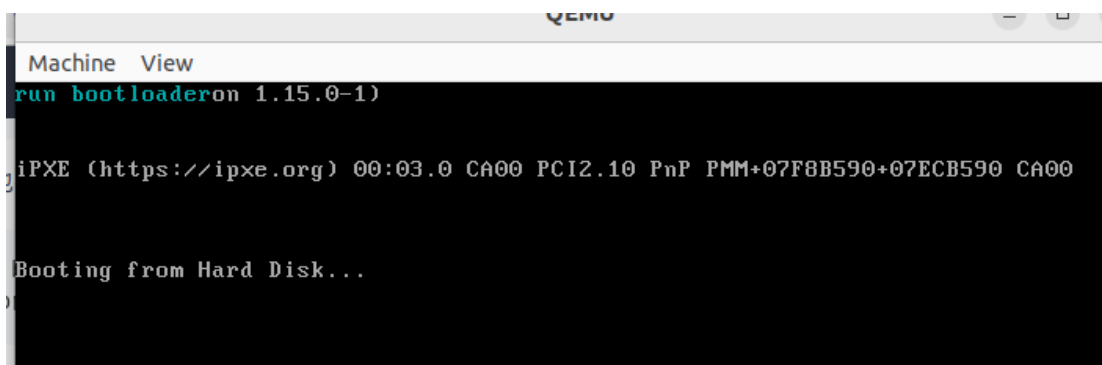
load_bootloader:
; 将LBA转换为CHS (公式: C=(LBA/SPT)/HPC, H=(LBA/SPT)%HPC,
S=(LBA%SPT)+1)
mov ax, si
xor dx, dx
mov di, 63        ; SPT=63
div di            ; AX=LBA/63, DX=LBA%63
mov cl, dl        ; 临时保存S=(LBA%63)

xor dx, dx
mov di, 18        ; HPC=18
div di            ; AX=(LBA/63)/18 (柱面), DX=(LBA/63)%18
(磁头)

; 组装CHS参数
inc cl            ; S=(LBA%63)+1 (扇区号1-based)
mov ch, al        ; 柱面号低8位
mov dh, dl        ; 磁头号
mov dl, 0x80      ; 驱动器号

```

3) 编译运行，得到和 example1 一样的结果



----- 实验任务 2 -----

- 任务要求：复现 example2，使用 gdb 或其他 debug 工具在进入保护模式的 4 个重要步骤上设置断点，并结合代码、寄存器的内容等来分析这 4 个步骤，最后附上结果截图。
- 思路分析：本实验的核心目标是在 bootloader 中实现从实模式到保护模式的转换，并在保护模式下输出提示信息。主要分为以下几个关键步骤：
 1. 内存规划：

- 1) MBR (0x7C00-0x7E00)
- 2) Bootloader (0x7E00-0x8800)
- 3) GDT (0x8800-0x8880)

2.保护模式切换流程:

- 1) 准备 GDT (全局描述符表)
- 2) 打开 A20 地址线
- 3) 设置 CR0 寄存器 PE 位
- 4) 远跳转进入保护模式

3.平坦模式设计:

- 1) 使用统一的 4GB 段描述符简化内存访问
- 2) 特殊处理显存段 (0xB8000)

4.代码改造:

- 1) 修改 MBR 的磁盘读取函数参数传递方式
- 2) 在 bootloader 中添加保护模式初始化代码

● 实验步骤:

1. 规划内存地址

Name	Start	Length	End
MBR	0x7c00	0x200(512B)	0x7e00
bootloader	0x7e00	0xa00(512B * 5)	0x8800
GDT	0x8800	0x80(8B * 16)	0x8880

2.将常量定义在 boot.inc 中

```

; 常量定义区
; _____Loader_____
; 加载器扇区数
LOADER_SECTOR_COUNT equ 5

```

```

; 加载器起始扇区
LOADER_START_SECTOR equ 1

; 加载器被加载地址
LOADER_START_ADDRESS equ 0x7e00

; _____GDT_____

; GDT 起始位置
GDT_START_ADDRESS equ 0x8800

```

2. 在 bootloader 中进入保护模式。使用 gdb 或其他 debug 工具在进入保护模式的 4 个重要步骤上设置断点，并结合代码、寄存器的内容等来分析这 4 个步骤。

1) 准备 GDT，用 lgdt 指令加载 GDTR 信息。

```

;空描述符
mov dword [GDT_START_ADDRESS+0x00],0x00
mov dword [GDT_START_ADDRESS+0x04],0x00

;创建描述符，这是一个数据段，对应0~4GB的线性地址空间
mov dword [GDT_START_ADDRESS+0x08],0x0000ffff ; 基地址为0，段界限为0xFFFFF
mov dword [GDT_START_ADDRESS+0x0c],0x00cf9200 ; 粒度为4KB，存储器段描述符

;建立保护模式下的堆栈段描述符
mov dword [GDT_START_ADDRESS+0x10],0x00000000 ; 基地址为0x00000000，界限0x0
mov dword [GDT_START_ADDRESS+0x14],0x00409600 ; 粒度为1个字节

;建立保护模式下的显存描述符
mov dword [GDT_START_ADDRESS+0x18],0x80007fff ; 基地址为0x000B8000，界限0x07FFF
mov dword [GDT_START_ADDRESS+0x1c],0x0040920b ; 粒度为字节

;创建保护模式下平坦模式代码段描述符
mov dword [GDT_START_ADDRESS+0x20],0x0000ffff ; 基地址为0，段界限为0xFFFFF
mov dword [GDT_START_ADDRESS+0x24],0x00cf9800 ; 粒度为4kb，代码段描述符

;初始化描述符表寄存器GDTR
mov word [pgdt], 39 ;描述符表的界限
lgdt [pgdt]

```

2) 打开第 21 根地址线。

```

in al,0x92 ;南桥芯片内的端口
or al,0000_0010B
out 0x92,al ;打开A20

```

3) 开启 cr0 的保护模式标志位。

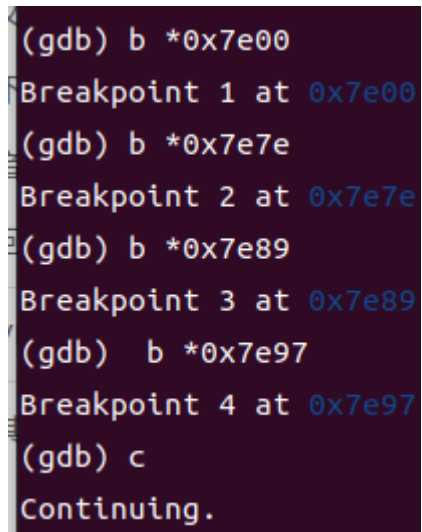
```
cli                                ;中断机制尚未工作
mov eax,cr0
or  eax,1
mov cr0,eax                       ;设置PE位
```

4) 远跳转，进入保护模式。

```
;以下进入保护模式
jmp dword CODE_SELECTOR:protect_mode_begin
```

gdb 调试:

1) 打上四个断点



```
(gdb) b *0x7e00
Breakpoint 1 at 0x7e00
(gdb) b *0x7e7e
Breakpoint 2 at 0x7e7e
(gdb) b *0x7e89
Breakpoint 3 at 0x7e89
(gdb) b *0x7e97
Breakpoint 4 at 0x7e97
(gdb) c
Continuing.
```

2) 第一个断点: 0x7e00

```
Breakpoint 1, 0x00007e00 in ?? ()
(gdb) info registers
eax            0x6                6
ecx            0x0                0
edx            0x80               128
ebx            0x8800             34816
esp            0x7c00             0x7c00
ebp            0x0                0x0
esi            0x0                0
edi            0x0                0
eip            0x7e00             0x7e00
eflags         0x286              [ IOPL=0 IF SF PF ]
cs             0x0                0
ss             0x0                0
ds             0x0                0
es             0x0                0
fs             0x0                0
gs             0x0                0
fs_base        0x0                0
gs_base        0x0                0
k_gs_base      0x0                0
cr0            0x10               [ ET ]
cr2            0x0                0
cr3            0x0                [ PDBR=0 PCID=0 ]
cr4            0x0                [ ]
```

3) 第二个断点: 0x7e7e


```
Breakpoint 2, 0x00007e7e in ?? ()
(gdb) info registers
eax            0x372            882
ecx            0x0             0
edx            0x80            128
ebx            0x1c            28
esp            0x7c00          0x7c00
ebp            0x0             0x0
esi            0x7eec          32492
edi            0x0             0
eip            0x7e7e          0x7e7e
eflags         0x202          [ IOPL=0 IF ]
cs             0x0             0
ss             0x0             0
ds             0x0             0
es             0x0             0
fs             0x0             0
gs             0xb800         47104
fs_base        0x0             0
gs_base        0xb8000        753664
k_gs_base      0x0             0
cr0            0x10           [ ET ]
cr2            0x0             0
cr3            0x0           [ PDBR=0 PCID=0 ]
cr4            0x0           [ ]
```

程序被正确装载。

4) 第三个断点: 0x7e89

```

Breakpoint 3, 0x00007e89 in ?? ()
(gdb) info registers
eax            0x372            882
ecx            0x0              0
edx            0x80             128
ebx            0x1c             28
esp            0x7c00           0x7c00
ebp            0x0              0x0
esi            0x7eec           32492
edi            0x0              0
eip            0x7e89           0x7e89
eflags         0x202            [ IOPL=0 IF ]
cs             0x0              0
ss             0x0              0
ds             0x0              0
es             0x0              0
fs             0x0              0
gs             0xb800           47104
fs_base        0x0              0
gs_base        0xb8000          753664
k_gs_base      0x0              0
cr0            0x10             [ ET ]
cr2            0x0              0
cr3            0x0              [ PDBR=0 PCID=0 ]
cr4            0x0              [ ]
--Type <RET> for more, q to quit, c to continue without paging--c
cr8            0x0              0
efer           0x0              [ ]

```

5) 第 4 个断点 0x7e97

```

Breakpoint 4, 0x00007e97 in ?? ()
(gdb) info registers
eax            0x11             17
ecx            0x0              0
edx            0x80             128
ebx            0x1c             28
esp            0x7c00           0x7c00
ebp            0x0              0x0
esi            0x7eec           32492
edi            0x0              0
eip            0x7e97           0x7e97
eflags         0x6              [ IOPL=0 PF ]
cs             0x0              0
ss             0x0              0
ds             0x0              0
es             0x0              0
fs             0x0              0
gs             0xb800           47104
fs_base        0x0              0
gs_base        0xb8000          753664
k_gs_base      0x0              0
cr0            0x10             [ ET ]
cr2            0x0              0
cr3            0x0              [ PDBR=0 PCID=0 ]
cr4            0x0              [ ]

```

最终结果：

```
Machine View
run bootloaderon 1.15.0-1)
enter protect mode

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+07F8B590+07ECB590 CA00

Booting from Hard Disk...
-
```

----- 实验任务 3 -----

- 任务要求：改造“Lab2-Assignment 4”为 32 位代码，即在加载到保护模式后执行汇编程序。
- 思路分析： 本实验通过 MBR 加载 Bootloader 到 0x7e00，切换到保护模式后，直接操作显存实现字符动画。核心步骤：
 1. MBR：用 LBA 模式加载 Bootloader；
 2. Bootloader：
 - 1) 初始化 GDT（含平坦模式段和显存段）；
 - 2) 开启 A20，设置 CR0.PE 进入保护模式；
 3. 动画逻辑：
 - 1) 计算显存偏移 $(y*80+x)*2$ ；
 - 2) 轮询字符和颜色，写入 gs:edi；
 - 3) 方向变量 d 控制移动路径，边界反弹。
 - 4) 全程绕过 BIOS 中断，直接硬件交互。

- 实验步骤：

1.写常量定义文件 boot.inc

```
1 ; 常量定义
2 LOADER_START_ADDRESS equ 0x7e00
3 GDT_START_ADDRESS equ 0x8800
4
5 ; 段选择子
6 DATA_SELECTOR equ 0x08
7 CODE_SELECTOR equ 0x10
8 VIDEO_SELECTOR equ 0x18
```

2. 写 mbr.asm, 负责加载 bootloader 到内存并且跳转

```
%include "boot.inc"

[bits 16]
org 0x7c00

start:
    xor ax, ax
    mov ds, ax
    mov es, ax
    mov ss, ax
    mov sp, 0x7c00

    ; 加载 bootloader
    mov ax, 1
    mov cx, 2
    mov bx, LOADER_START_ADDRESS

load_loop:
    push ax
    push bx
    call read_disk
    add sp, 4
    inc ax
    add bx, 512
    loop load_loop

    jmp 0:LOADER_START_ADDRESS

read_disk:
    push bp
    mov bp, sp
    pusha

    mov ax, [bp+6]
    mov dx, 0x1f3
    out dx, al
    inc dx
    mov al, ah
    out dx, al
    xor ax, ax
    inc dx
    out dx, al
    inc dx
    or al, 0xe0
    out dx, al

    mov dx, 0x1f2
    mov al, 1
    out dx, al
    mov dx, 0x1f7
    mov al, 0x20
    out dx, al

.wait:
    in al, dx
    and al, 0x88
```

```

    cmp al, 0x08
    jnz .wait

    mov bx, [bp+4]
    mov cx, 256
    mov dx, 0x1f0
.read:
    in ax, dx
    mov [bx], ax
    add bx, 2
    loop .read

    popa
    pop bp
    ret

times 510-($-$$) db 0
dw 0xaa55

```

3. 把原先的代码写入 bootloader.asm 中，并做修改。

```

#include "boot.inc"

[bits 16]
org 0x7e00

start:
    ; 初始化段寄存器
    xor ax, ax
    mov ds, ax
    mov es, ax
    mov ss, ax
    mov sp, 0x7c00

    ; 设置 GDT
    mov dword [GDT_START_ADDRESS+0x00], 0x00000000 ; 空描述符
    mov dword [GDT_START_ADDRESS+0x04], 0x00000000

    ; 平坦模式数据段
    mov dword [GDT_START_ADDRESS+0x08], 0x0000ffff
    mov dword [GDT_START_ADDRESS+0x0c], 0x00cf9200

    ; 平坦模式代码段
    mov dword [GDT_START_ADDRESS+0x10], 0x0000ffff
    mov dword [GDT_START_ADDRESS+0x14], 0x00cf9800

    ; 显存段 (0xB8000)
    mov dword [GDT_START_ADDRESS+0x18], 0x8000ffff
    mov dword [GDT_START_ADDRESS+0x1c], 0x0040920b

    ; 加载 GDTR
    mov word [gdt_ptr], 31
    mov dword [gdt_ptr+2], GDT_START_ADDRESS
    lgdt [gdt_ptr]

    ; 开启 A20

```

```

    in al, 0x92
    or al, 2
    out 0x92, al

; 切换到保护模式
cli
mov eax, cr0
or eax, 1
mov cr0, eax

jmp dword CODE_SELECTOR:protected_mode

[bits 32]
protected_mode:
; 初始化段寄存器 (32 位)
mov ax, DATA_SELECTOR
mov ds, ax
mov es, ax
mov fs, ax
mov ax, VIDEO_SELECTOR
mov gs, ax
mov ss, ax
mov esp, 0x7c00

; 初始化变量
mov byte [x], 0
mov byte [y], 0
mov byte [d], 0
mov dword [c], 0
mov dword [s], 0

main_loop:
; 方向判断
cmp byte [d], 0
je go_right
cmp byte [d], 1
je go_down
cmp byte [d], 2
je go_left
cmp byte [d], 3
je go_up

go_right:
inc byte [x]
cmp byte [x], 79
jb show
mov byte [d], 1
dec byte [x]
jmp show

%include "boot.inc"

[bits 16]
org 0x7e00

start:
; 初始化段寄存器
xor ax, ax

```

```

mov ds, ax
mov es, ax
mov ss, ax
mov sp, 0x7c00

; 设置 GDT
mov dword [GDT_START_ADDRESS+0x00], 0x00000000 ; 空描述符
mov dword [GDT_START_ADDRESS+0x04], 0x00000000

; 平坦模式数据段
mov dword [GDT_START_ADDRESS+0x08], 0x0000ffff
mov dword [GDT_START_ADDRESS+0x0c], 0x00cf9200

; 平坦模式代码段
mov dword [GDT_START_ADDRESS+0x10], 0x0000ffff
mov dword [GDT_START_ADDRESS+0x14], 0x00cf9800

; 显存段 (0xB8000)
mov dword [GDT_START_ADDRESS+0x18], 0x8000ffff
mov dword [GDT_START_ADDRESS+0x1c], 0x0040920b

; 加载 GDTR
mov word [gdt_ptr], 31
mov dword [gdt_ptr+2], GDT_START_ADDRESS
lgdt [gdt_ptr]

; 开启 A20
in al, 0x92
or al, 2
out 0x92, al

; 切换到保护模式
cli
mov eax, cr0
or eax, 1
mov cr0, eax

jmp dword CODE_SELECTOR:protected_mode

```

[bits 32]

protected_mode:

```

; 初始化段寄存器 (32 位)
mov ax, DATA_SELECTOR
mov ds, ax
mov es, ax
mov fs, ax
mov ax, VIDEO_SELECTOR
mov gs, ax
mov ss, ax
mov esp, 0x7c00

; 初始化变量
mov byte [x], 0
mov byte [y], 0
mov byte [d], 0
mov dword [c], 0
mov dword [s], 0

```

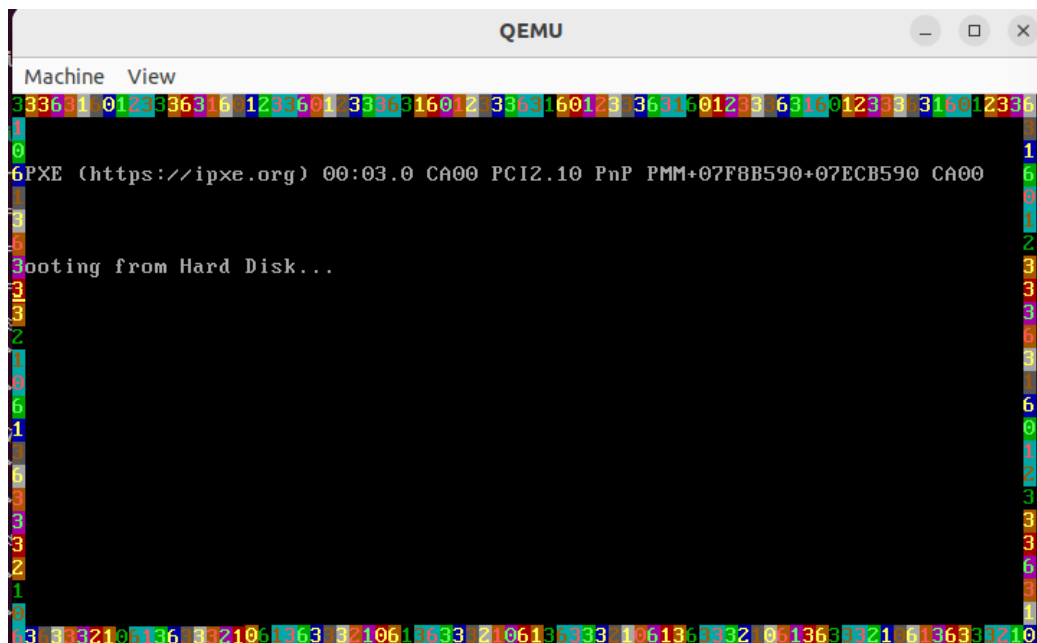
```

main_loop:
    ; 方向判断
    cmp byte [d], 0
    je go_right
    cmp byte [d], 1
    je go_down
    cmp byte [d], 2
    je go_left
    cmp byte [d], 3
    je go_up

go_right:
    inc byte [x]
    cmp byte [x], 79
    jb show
    mov byte [d], 1
    dec byte [x]
    jmp show

```

- 实验结果展示：通过执行前述代码，可得下图结果。



Section 5 实验总结与心得体会

这次实验较以前的简单不少，感恩感恩。学会了 LBA 方式下和 CHS 方式下的磁盘读取方式，学习了由实模式向保护模式的转换，更加理解了怎么利用 gdb 来调试等，受益匪浅。